Discrete-to-Deep Supervised Policy Learning

An effective training method for neural reinforcement learning

Paper #1129 with revision

ABSTRACT

Neural networks are popular and effective function approximators. However, in the reinforcement learning (RL) context, neural networks are hard to train mainly because samples are correlated. For years, scholars have got around this by employing experience replay, which makes learning very slow, or an asynchronous parallel-agent system, which requires complex infrastructure. This paper proposes Discrete-to-Deep Supervised Policy Learning (D2D-SPL), a variant of supervised policy learning, for training neural networks in RL. D2D-SPL discretises the continuous state space into discrete states and uses actor-critic to learn a policy. It then selects from each discrete state an input value and the action with the highest numerical preference for the discrete state as an input/target pair. Finally it uses input/target pairs from all discrete states to train a classifier. D2D-SPL uses a single agent, needs no experience replay and learns much faster than state-of-the-art methods. We test our method with two RL environments, the classic Cartpole and an air combat simulator.

KEYWORDS

reinforcement learning; neural networks; actor-critic; air-combat simulation

1 INTRODUCTION

2 RELATED WORK

3 REINFORCEMENT LEARNING BACKGROUND

4 DISCRETE-TO-DEEP SUPERVISED POLICY LEARNING (D2D-SPL)

D2D-SPL uses ...(removed for brevity). Note that *create_classifier* has been renamed *create_training_set*.

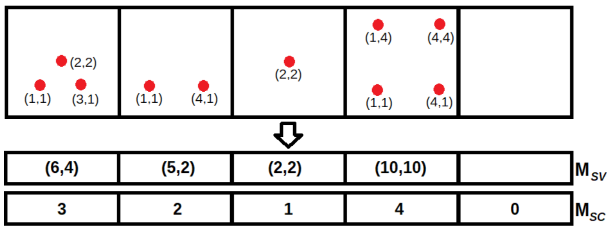

Figure 1: Consolidating data every episode

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 2020, Auckland, New Zealand

https://doi.org/doi

Figure 1 shows how data is consolidated every episode. For simplicity it shows a case where there are only five discrete states (each represented by a box at the top diagram) and there are two state variables in each state. At every timestep, the values of the state variables of the visited state are recorded. Figure 1 shows the data collected after ten timesteps, in which Discrete State 1 has been visited three times with state variable values (1,1), (2,2) and (3,1), Discrete State 2 visited twice with state variable values (1,1) and (4,1) and so on. Since we are only concerned with the average values of state variables in each discrete state, we can save memory by just keeping the totals of state variable values in every discrete state in M_{SV} and the number of times that state is visited in M_{SC}. Along with the total reward for the episode, we insert the tuple $(R_{TOTAL}, M_{SV}, M_{SV})$ to buffer B. At the end of Algorithm 1, the number of tuples in buffer B will be the same as the number of episodes run.

Buffer B and the actor-critic policy are passed to function *create_training_set* in Algorithm 2. The buffer contains the fore-mentioned tuples and the policy contains numerical preferences (one for each action) for each discrete state. The objective of this function is to produce an input/target pair for every discrete state, excluding states that were never visited during training. The function starts by selecting the top 5% tuples with the highest total rewards in the buffer and consolidate the selected tuples into at most one tuple for each discrete state. Data is consolidated by summing the values of each state variable in a discrete state and divided them by the number of times the state was visited. For each discrete state, we look up the policy and select as a target the action with the highest numerical preference. We remove all states that are not present in the input set from the target set and pass the training set to the classifier.

5 EXPERIMENTS AND DISCUSSION

We test our method with two RL environments, the classic Cartpole and an air combat simulator. We choose Cartpole because it is a well-known problem and the air combat simulator because it represents a large-scale problem that is not easy to solve using table-based methods.

In each of the two problems, the agent learns an actor-critic policy for n episodes. We call the policy it learns Discrete n and use it as the base for several other solutions. In the first solution we clone the policy and use the clone as the base policy to run another n episodes of actor-critic learning and call the resulting policy Discrete $2n$, because it is the result of learning for $2n$ episodes. In the second solution, which we call D2D-SPL, we use the data generated from Discrete n to train a classifier.

Separately, for comparison, we use DQN [5], Double DQN (DDQN) [6] and A3C [4] to solve the same problems. A3C, despite it being four years old, is currently the state-of-the-art method. We use four parallel agents in all our A3C tests.

For Cartpole, we use eight systems to compare: Discrete 1,000, Discrete 2,000, D2D-SPL, DQN 1,000, DQN 2,000, DDQN 1,000, DDQN 2,000 and A3C 4,000. For Air Combat we use Discrete 20,000, Discrete 40,000, D2D-SPL, DQN 20,000, DQN 40,000, DDQN 20,000, DDQN 40,000, A3C 20,000 and A3C 200,000. All experiments are run on an Intel i9-7900X (10 cores, 20 threads) machine with two GTX 10 GPU cards. For all the methods involving neural networks, we use a relatively simple architecture containing a single hidden layer with twelve nodes in Cartpole and fifty nodes in Air Combat.

5.1 Cartpole

Cartpole is a pole-balancing control problem posed in [3] for which Barto et. al. proposed a solution using actor-critic reinforcement learning [1]. A solution aims to control the free-moving cart to which the pole is attached by exerting a force to the left or to the right of the cart to keep the pole standing. We use the OpenAI Gym Cartpole environment [2] that is based on Barto et. al.'s solution but is different from the original system in that the four state variables in Gym are randomised at the beginning of every episode, whereas in the latter the variables are always set to zero.

OpenAI Gym's Cartpole comes in two flavours, one that considers the problem solved if the pole remains standing for 200 consecutive timesteps and another whose target is 500 timesteps. We make the problem more complex by raising the target to 100,000.

Our code is published on https://github.com/author-one/d2d-spl.

The agent gets a +1 reward for each timestep, including when the agent lands on a terminal state. Each episode is terminated when one of these two things occurs: When the pole falls (failure) or when the target is achieved (success). Therefore, the time spent in each episode is proportional to the number of total rewards. The more successful learning in an episode is, the longer the episode takes.

Table 1 shows the learning times for all solutions for ten trials. All numbers are relative to the average learning time of Discrete 1,000. On average, Discrete 2,000 took almost six times longer than Discrete 1,000. The reason for this is that after 2,000 episodes the policies are better at maintaining the pole and therefore the later episodes take longer to complete than the early ones. Like Discrete 2,000, D2D-SPL is also based on Discrete 1,000 but learns much faster as it only takes 4% longer than Discrete 1,000.

All the DQN solutions take much longer to learn, indicating much slower learning than actor-critic and D2D-SPL. Our A3C solution, which uses four parallel agents, is faster than DQN and DDQN, completing 4,000 episodes in twice the time taken by D2D-SPL. We train our A3C solution in 4,000 episodes because tests with fewer episodes resulted in very little learning.

Using the policies of Discrete 1,000 and Discrete 2,000 and the models of D2D-SPL, (D)DQN and A3C, we run ten trials that each consists of 100 independent runs and sum the total rewards for each trial. Each run is unique due to the fact that the starting state variables are random. We use the same random seeds for all methods, making sure the initial values for all methods are the same for each trial. The average rewards for all runs are shown in Table 2.

Table 3 shows how many tests in each trial achieve the target reward of 100,000.

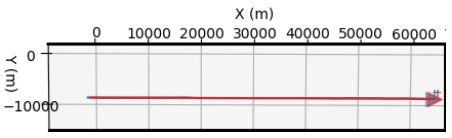

Figure 2: Air-combat test scenario 1

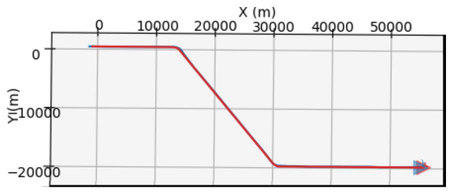

Figure 3: Air-combat test scenario 2

Table 1 shows that D2D-SPL learns much faster than the compared methods. Tables 2 and 3 show that D2D-SPL is more effective than all the other methods, except DDQN. A3C, even though it learns faster thanks to multiple agents running in parallel, performs much worse than the other solutions.

5.2 Air Combat Manoeuvring Simulator

5.2.1 *The Environment.*

5.2.2 *Test Results.* For the actor-critic solutions, we discretise the state space into 14,000 discrete states. The range is split into fourteen regions, the AA into ten regions, the ATA into ten regions, and the speed difference into ten regions, resulting in 14,000 states.

For training for all solutions, we start the opponent (Red aircraft) from position (x_r, y_r, ψ_r) where x_r and y_r are a coordinate in a Cartesian coordinate and ψ_r the flying direction (heading) in degrees (relative to the X axis). Our aircraft (Blue) always starts from the origin with heading 0°and an initial speed of 125m/s, which means it starts by flying along the X axis. Red always starts from (1500±Δ, 300±Δ, 50°±Δ), where Δ is a small positive random number, and flies in a straight line at a constant speed of 125m/s. The initial positions, headings and speeds of both aircraft are the same for all episodes. All episodes are terminated after 700 timesteps.

We start by running the agent for 20,000 episodes, resulting in Discrete 20,000. This base policy is then cloned for the same agent to continue learning another 20,000 agents, resulting in Discrete 40,000. The base is also used for D2D-SPL. Separately, we use DQN, DDQN and A3C to produce DQN 20,000, DQN 40,000, DDQN 20,000, DDQN 40,000, A3C 20,000 and A3C 200,000. The choice for 200,000 episodes for the second A3C solution is due to the fact that A3C is data inefficient and need much more episodes to achieve comparable scores.

The policies from Discrete 20,000 and Discrete 40,000 as well as models from D2D-SPL, DQN, DDQN and A3C solutions are then used to test agents against an opponent that flies along paths that were not seen during training. Figures 2 to 5 show four test scenarios. The red paths represent the opponent's trajectories and the blue ones our agent's.

Table 4 shows the learning times for all the solutions. Due to the fact that each episode is terminated after a fixed number of

Trial	Discrete	Discrete	**D2D**	DQN	DQN	DDQN	DDQN	A3C
	1,000	2,000	SPL	1,000	2,000	1,000	2,000	4,000
0	1.77	8.37	**1.82**	24.80	44.99	67.91	562.73	1.11
1	0.72	1.57	**0.76**	23.36	59.09	46.10	416.00	1.38
2	0.59	1.40	**0.64**	4.95	25.30	15.57	435.13	3.13
3	1.23	2.68	**1.28**	17.80	35.42	95.83	597.61	1.20
4	0.95	2.12	**0.99**	14.97	33.43	30.66	646.35	1.73
5	0.46	0.96	**0.47**	20.63	41.33	140.80	773.70	1.68
6	2.03	34.71	**2.09**	30.36	159.95	49.23	496.53	3.36
7	0.92	2.08	**0.99**	37.65	109.29	58.71	708.72	2.23
8	0.44	0.90	**0.47**	20.37	47.73	85.27	509.30	1.21
9	0.87	2.17	**0.90**	36.34	75.40	45.64	459.48	3.97
Mean	1.00	5.70	**1.04**	23.12	63.19	63.57	560.55	2.10

Table 1: Cartpole learning times relative to the mean of Discrete 1,000

Trial	Discrete	Discrete	D2D	DQN	DQN	DDQN	DDQN	A3C
	1,000	2,000	SPL	1,000	2,000	1,000	2,000	4,000
0	443.42	2395.40	**41,205.76**	9.31	46.57	426.51	13.78	17.83
1	253.01	443.85	**636.60**	70.61	1,476.75	69049.44	163.32	24.47
2	378.59	495.04	**96,004.04**	9.48	116.52	83,064.61	100,000.00	93.93
3	241.19	208.69	**32.01**	11.47	30.17	143.71	78,132.04	12.02
4	206.49	210.14	**14,194.83**	16.09	10.80	100,000.00	288.71	76.05
5	130.79	144.49	**18,623.59**	28.81	30.61	9.71	243.64	10.32
6	2,177.08	2,581.67	**1,098.02**	207.67	9.42	18.02	446.12	12.90
7	349.99	230.72	**100,000.00**	178.64	69.46	92.76	69.40	14.34
8	137.32	112.44	**86.46**	437.77	23.53	100,000.00	191.21	9.47
9	349.23	363.38	**12.28**	7,160.00	11.14	89.90	189.81	119.22
Mean	466.71	718.58	**27189.36**	812.99	182.50	35,289.47	17,973.80	39.06
Median	301.12	297.05	**7,646.43**	49.71	30.39	285.11	217.43	16.09

Table 2: Average rewards for all Cartpole solutions

Trial	Discrete	Discrete	D2D	DQN	DQN	DDQN	DDQN	A3C
	1,000	2,000	SPL	1,000	2,000	1,000	2,000	4,000
0	0	0	**41**	0	0	0	0	0
1	0	0	**0**	0	1	69	0	0
2	0	0	**96**	0	0	83	100	0
3	0	0	**0**	0	0	0	78	0
4	0	0	**14**	0	0	100	0	0
5	0	0	**6**	0	0	0	0	0
6	0	0	**1**	0	0	0	0	0
7	0	0	**100**	0	0	0	0	0
8	0	0	**0**	0	0	100	0	0
9	0	0	**0**	7	0	0	0	0

Table 3: Successes in solving the Cartpole problem

timesteps, the learning times are more predictable. For example, Discrete 40,000 takes about twice the time taken by Discrete 20,000, DQN 40,000 runs in about twice the time taken by DQN 20,000, and DDQN 40,000 completes in twice the time taken by DDQN 20,000. A3C 200,000 runs ten times longer than A3C 20,000. Because A3C uses four concurrent agents, learning takes much faster than its DQN and DDQN rivals. Among all the solutions being compared, D2D-SPL learns the fastest as it only takes 0.01% longer than Discrete 20,000.

Figure 6 shows the average reward per episode for all the solutions. The graphs have been smoothed-out by replacing every 200 consecutive rewards with their mean. It shows that using data from Discrete 20,000 to train the D2D-SPL network results in an average score of 0.95 when the training set is re-applied to the resulting

Trial	Discrete 20,000	Discrete 40,000	D2D SPL	DQN 20,000	DQN 40,000	DDQN 20,000	DDQN 40,000	A3C 20,000	A3C 200,000
0	0.9515	1.9079	**0.9515**	4.6971	11.1307	3.5559	7.2530	1.3936	13.5347
1	0.9476	1.8884	**0.9476**	4.8871	12.2939	3.4940	7.2707	1.3605	13.5817
2	1.1920	2.3880	**1.1923**	4.4365	13.8925	3.5478	7.2975	1.2748	13.4719
3	1.1778	2.3617	**1.1779**	4.5109	12.3102	4.4111	8.7122	1.2184	14.3311
4	1.2071	2.4377	**1.2072**	3.5728	7.3820	4.2812	8.3312	1.2382	14.0830
5	0.9106	1.8422	**0.9106**	3.5866	7.3969	4.1718	8.2209	1.3658	14.5852
6	0.9104	1.8391	**0.9104**	3.5898	7.3570	3.8600	7.7078	1.4976	13.8962
7	0.9127	1.8497	**0.9127**	3.5889	7.3631	3.7903	7.6311	1.5488	13.9328
8	0.8952	1.8074	**0.8952**	3.9266	8.9514	3.8597	7.8547	1.3989	14.9915
9	0.8952	1.8004	**0.8952**	3.8923	8.6911	3.8398	7.9101	1.3100	13.7658
Mean	1.0000	2.0123	**1.0001**	4.0688	9.6769	3.8812	7.8189	1.3607	14.0174

Table 4: Air combat learning times relative to the mean of Discrete 20,000

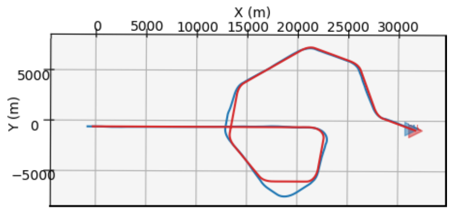

Figure 4: Air-combat test scenario 3

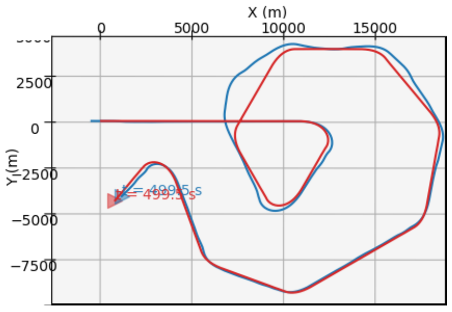

Figure 5: Air-combat test scenario 4

network. This score is much higher than the scores of the other methods.

It can also seen that DQN suffers from overestimation as reported in [6], which adversely affects the policy. Overestimation did not occur in DDQN.

Tables 5 to 8 show the results for the four test scenarios. In the discrete cases, Discrete 40,000 is generally better than Discrete 20,000 because the learning curve with actor-critic is more stable, not as noisy as the DQN or A3C. This means, longer learning tends to produce a better policy. In all cases, D2D-SPL performs better than its base Discrete 20,000 and even Discrete 40,000. It is also better than DQN 20,000 in three of four cases and better than DQN 40,000 in all cases. Because of the possible overestimation in DQN, there is no guarantee that longer learning (in this case, DQN 40,000) will produce a better model than shorter learning (in this case, DQN 20,000).

6 CONCLUSIONS AND FUTURE WORK

We build and test solutions for Cartpole and an air combat simulator. The difference among the solutions is the learning time and the performance of the generated policy or model. It is shown that actor-critic works for both problems and longer learning with more episodes tends to produce a better policy, even though, as shown in Figure 6, the result is unstable in the sense that the policy after n episodes is not always better than the policy before that. This means, when we decide to stop learning after episode n, we need to record m policies before episode n, test them against some pre-set criteria, and choose the best policy. For instance, if we decide to run a learning session for 20,000 episodes, we might want to compare all policies resulting from the last 500 episodes or so.

We also show that D2D-SPL can shorten the learning time of the actor-critic algorithm. The D2D-SPL results are consistently better than the base policy used to train it and even better than the policy obtained by resuming the actor-critic learning. In addition, because D2D-SPL gets its data from the top 5% of the episodes, the resulting model is stable. As D2D-SPL uses a neural-network, which generally is known to be a good function approximator, it is not surprising that D2D-SPL performs better than discrete actor-critic in generalising test cases not seen during learning.

In both Cartpole and Air Combat, DQN, DDQN and A3C can be used to train a neural network. However, D2D-SPL learns much faster and performs better in Cartpole and in the majority of the test cases in Air Combat than its competitors.

One difficulty in using D2D-SPL is to find a discretisation scheme that leads to a good policy. In the case of Cartpole, the discretisation scheme has been made available in [1]. In the case of Air Combat it took us many experiments to come up with a good discretisation scheme. Generally, the more state variables there are, the more combinations there are that are possible and the harder to get it right. Future studies may focus on using D2D-SPL in environments with higher numbers of state variables.

Since this is the first time D2D-SPL is ever used, there is room for improvement. Future work may utilise other tabular RL algorithms, such as Q-learning and SARSA in the RL part of the system. It is also possible to further train the resulting network of D2D-SPL, using an existing or new method, to improve performance.

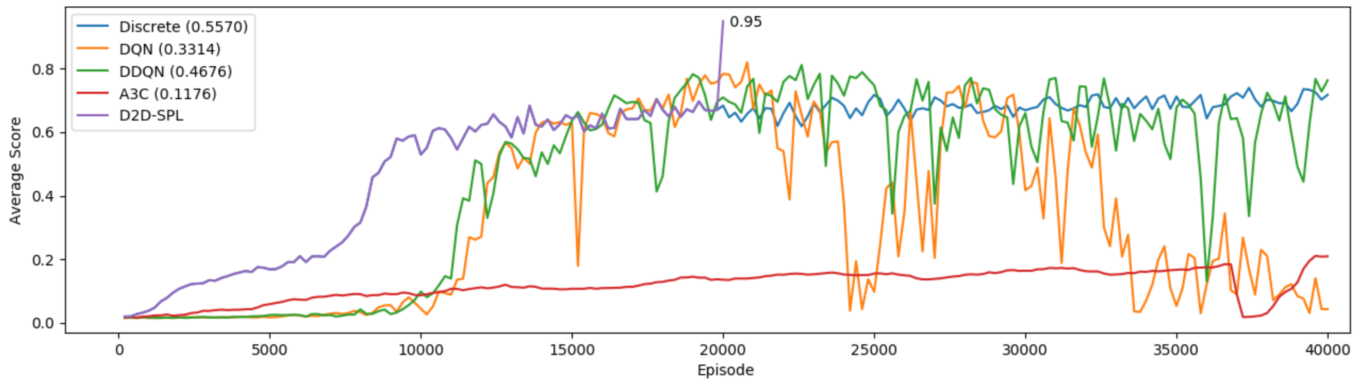

Figure 6: Reward/episode for aircombat for all solutions (200:1)

Trial	Discrete 20,000	Discrete 40,000	D2D SPL	DQN 20,000	DQN 40,000	DDQN 20,000	DDQN 40,000	A3C 20,000	A3C 200,000
0	0.9123	0.9182	**0.9502**	0.9065	0.0259	0.7816	0.5363	0.0478	0.2515
1	0.8142	0.8142	**0.8956**	0.8748	0.4256	0.0558	0.1104	0.1430	0.0485
2	0.8508	0.9130	**0.7656**	0.2960	0.0327	0.0755	0.4394	0.1519	0.1575
3	0.9274	0.9236	**0.9312**	0.0250	0.0327	0.9777	0.5637	0.1611	0.0478
4	0.9086	0.9205	**0.9578**	0.8620	0.9627	0.7788	0.4392	0.1638	0.0813
5	0.9899	0.9899	**0.9572**	0.7936	0.0286	0.7941	0.8102	0.1681	0.8710
6	0.8142	0.8142	**0.9490**	0.9687	0.0256	0.7283	0.2349	0.1599	0.8748
7	0.9237	0.9204	**0.9530**	0.7928	0.1620	0.9821	0.4543	0.1689	0.1026
8	0.8142	0.8142	**0.9007**	0.0412	0.1170	0.8645	0.3903	0.2033	0.2055
9	0.8142	0.8142	**0.9463**	0.9211	0.8751	0.8483	0.2347	0.0478	0.1298
Mean	0.8769	0.8842	**0.9207**	0.6482	0.2688	0.6887	0.4213	0.1416	0.2770

Table 5: Air combat test results for test scenario 1

Trial	Discrete 20,000	Discrete 40,000	D2D SPL	DQN 20,000	DQN 40,000	DDQN 20,000	DDQN 40,000	A3C 20,000	A3C 200,000
0	0.9107	0.6160	**0.9502**	0.8564	0.0259	0.7838	0.3568	0.0478	0.2372
1	0.7999	0.7999	**0.8956**	0.3352	0.3411	0.0500	0.0567	0.1689	0.0478
2	0.7007	0.8771	**0.7656**	0.0806	0.0327	0.0762	0.4437	0.1623	0.1306
3	0.8747	0.8329	**0.9312**	0.0250	0.0327	0.3423	0.7283	0.1779	0.0478
4	0.6414	0.8424	**0.9578**	0.8459	0.9580	0.6630	0.6578	0.1618	0.0800
5	0.9054	0.8895	**0.9572**	0.7894	0.0286	0.7204	0.7568	0.1418	0.8560
6	0.7999	0.7999	**0.9490**	0.9376	0.0256	0.8829	0.8234	0.2059	0.2092
7	0.8858	0.8662	**0.9530**	0.7832	0.1571	0.8347	0.7588	0.1616	0.1333
8	0.7999	0.7999	**0.9007**	0.0412	0.1272	0.8624	0.7429	0.2410	0.1427
9	0.8559	0.8458	**0.9463**	0.3516	0.8442	0.7489	0.7484	0.0478	0.1919
Mean	0.8174	0.8170	**0.9207**	0.5046	0.2573	0.5965	0.6074	0.1517	0.2077

Table 6: Air combat test results for test scenario 2

Trial	Discrete 20,000	Discrete 40,000	D2D SPL	DQN 20,000	DQN 40,000	DDQN 20,000	DDQN 40,000	A3C 20,000	A3C 200,000
0	0.3760	0.3463	**0.8334**	0.8378	0.0259	0.6751	0.2139	0.0478	0.2988
1	0.4326	0.7685	**0.7123**	0.4427	0.1853	0.0558	0.1496	0.1755	0.0478
2	0.7350	0.7476	**0.2928**	0.2056	0.0327	0.0755	0.5904	0.1607	0.0556
3	0.7490	0.6128	**0.7393**	0.0250	0.0327	0.7820	0.5738	0.1756	0.0478
4	0.4257	0.4731	**0.8563**	0.8399	0.9223	0.8954	0.5463	0.1806	0.0816
5	0.4436	0.4917	**0.3530**	0.7523	0.0286	0.6745	0.4392	0.1585	0.6567
6	0.6452	0.3009	**0.4664**	0.8984	0.0256	0.8765	0.5592	0.1778	0.3118
7	0.3349	0.6714	**0.6665**	0.7423	0.1576	0.4637	0.5429	0.1702	0.1271
8	0.3562	0.3962	**0.3290**	0.0412	0.0852	0.3291	0.8912	0.2594	0.1800
9	0.3893	0.4662	**0.3332**	0.3484	0.7212	0.4329	0.8159	0.0478	0.1912
Mean	0.4887	0.5275	**0.5582**	0.5133	0.2217	0.5260	0.5322	0.1554	0.1998

Table 7: Air combat test results for test scenario 3

Trial	Discrete 20,000	Discrete 40,000	D2D SPL	DQN 20,000	DQN 40,000	DDQN 20,000	DDQN 40,000	A3C 20,000	A3C 200,000
0	0.5641	0.3044	**0.8093**	0.8006	0.0283	0.2382	0.1329	0.0478	0.2773
1	0.2615	0.3348	**0.3915**	0.2863	0.1389	0.0501	0.0506	0.1758	0.0478
2	0.1841	0.4393	**0.1782**	0.0774	0.0327	0.0949	0.5106	0.1938	0.1976
3	0.1992	0.2574	**0.1892**	0.0250	0.0327	0.4205	0.4329	0.1844	0.0478
4	0.1859	0.1948	**0.8340**	0.8197	0.6820	0.4563	0.2341	0.2020	0.0789
5	0.1987	0.3113	**0.1994**	0.7693	0.0371	0.5742	0.4293	0.1384	0.6853
6	0.2172	0.5787	**0.2083**	0.8142	0.0256	0.5667	0.4789	0.1915	0.1717
7	0.1933	0.2045	**0.7529**	0.8031	0.1510	0.4172	0.5499	0.1637	0.1226
8	0.1744	0.6813	**0.1978**	0.0412	0.0858	0.4810	0.5519	0.2881	0.2561
9	0.1808	0.2318	**0.2194**	0.2001	0.5020	0.6722	0.6991	0.0478	0.1654
Mean	0.2359	0.3538	**0.3980**	0.4637	0.1716	0.3971	0.4070	0.1633	0.2050

Table 8: Air combat test results for test scenario 4

REFERENCES

[1] A. Barto, R.S. Sutton, and C.W Anderson. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on System, Man, and Cybernetics* (1983), 833–836.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J Schulman, J. Tang, and W. Zaremba. 2016. OpenAI Gym. (2016).

[3] D. Michie and R.A. Chambers. 1968. BOXES: An experiment in adaptive control. *Machine Intelligence 2, E. Dale and D. Michie, Eds. Edinburgh: Oliver and Boyd* 2 (1968), 137–152.

[4] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Harley, T.P. Lillicrap, D. Silver, and K. Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. *Proc. 33rd Int. Conf. Mach. Learn.* 48 (2016).

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. Playing Atari with deep reinforcement learning. *NIPS Deep Learning Workshop* (2013).

[6] H. van Hasselt, A. Guez, and D. Siver. 2016. Deep reinforcement learning with double Q-learning. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (2016).